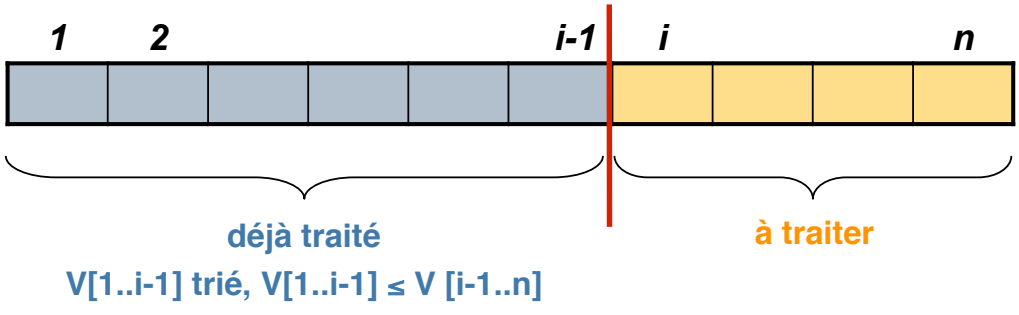
Algorithmique & programmation

Chapitre 2 : Vecteurs

Algorithme de tri par permuation

Tri par permutation

- Construction de l'hypothèse
  - On est en train de trier le vecteur
  - Donc on va partir de l'hypothèse que l'on a déjà trié une partie du vecteur



Tri par permutation

- Idée de l'algorithme
  - Dans la partie du vecteur non traitée (V[i..n])
    - Chercher l'indice du plus petit élément
    - Permuter cet élément avec V[i]
  - Ainsi
    - V[1..i] sera trié
  - Poursuivre jusqu'à ce que le vecteur non traité devienne vide

Tri par permutation

- Raisonnement par récurrence

Hypothèse V[1..i-1] trié, V[1..i-1] ≤ V [i-1..n] , V[i..n] non traité

➤ i = n ⇨ *V[1..n-1] est trié, V[1..n-1] ≤ V[n]*
⇨ V[1..n] est trié ✳

➤ i < n

soit j tel que V[j] = **minimum** (V[i..n])

➤➤ j = i ⇨ i := i + 1 ; ➡ ***H***

➤➤ j ≠ i ⇨ **permuter** V[j] et V[i] ; i := i + 1 ; ➡ ***H***

Itération tantque (i < n) faire …

Initialisation i := 1 ; ➡ ***H***

procédure **tripermut**

procédure tripermut (**dr** V[1..n] : vecteur) ;
spécification *{}* ➔ *{V[1..n] trié}*
    i, j : entier ;
debproc
    i := 1 ;
    tantque i < n **faire**
        j := **indmin** (V[i..n]) ;
        si j ≠ i alors
            permut (V[i], V[j]) ;
        finsi ;
        i := i + 1 ;
    finfaire ;
finproc ;

procédure **tripermut**

- soit V un vecteur d'entier

```
procedure tripermut (V : in out vecteur) is
--spécification {} ➔ {V[1..n] trié}
  i, j : integer ;
begin
  i := V'First ;
  while i < V'Last loop
    j := indmin (V(i..n)) ;
    if j /= i then
      permut (V(i), V(j)) ;
    end if ;
    i := i + 1 ;
  end loop ;
end tripermut ;
```

fonction **indmin**

fonction indmin (d V[**i**..n] : vecteur) : entier ;
spécification *{1≤i<n} ➔ {résultat = j, j ∈ [i..n], V[j] ≤ V[i..n]}*
    k, j : entier ;
debfonc
    j := i ;
    k := i + 1 ;
    tantque k ≤ n **faire**
        si V[j] > V[k] **alors**
            j := k ;
        finsi ;
        k := k + 1 ;
    finfaire ;
    retour k ;
finfonc ;

fonction **indmin**

- soit V un vecteur d'entier

```
function indmin (V : in vecteur) return integer is
--spec {1≤i<n}➔{résultat=j, j∈[i..n], V[j]≤V[i..n]}
  k, j : integer ;
begin
  j := V'First ;
  k := V'First + 1 ;
  while k ≤ V'Last loop
    if V(j) > V(k)) then
      j := k ;
    end if ;
    k := k + 1 ;
  end loop ;
  return k ;
end indmin ;
```

Coût de **tripermut**

- Place occupée
  - 1 vecteur + 1 variable pour la permutation
  - **(n + 1) t**
- Coût de **permut**
  - 3 affectations (4 accès)
- Coût de **indmin** (m est la taille du vecteur)
  - 1 comparaison à chaque itération
  - m - 1 comparaisons (2 (m - 1) accès)

Coût de **tripermut**

- Nombre de comparaisons
  - À chaque itération appel de **indmin** (V[i..n]) avec i qui varie de 1 à n - 1
  - = n - 1 + n - 2 + … + n - j + … + 2 + 1
  - $= \sum_{j=1}^{n-1} j$ = **n(n - 1) / 2** (n(n - 1) accès, 2 / comp.)
- Nombre de permutations
  - Cas favorable (vecteur trié), **permut** non appelée
  - 0 permutation
    - Cas défavorable (trié ordre inverse), **permut** à chaque itération
  - n - 1 permutations (4 (n - 1) accès)

Coût de **tripermut**

- Pour résumer

place occupée		(n + 1)t	
nombre de comparaisons		n(n - 1)/2	(≈ n^2 accès)
nombre d'affectations	cas favorable	0	
	cas défavorable	n - 1	(≈ 4n accès)

- Exemple
  - Si n = 1000 et t = 20 octets
  - Place occupée : 20 020 octets
  - nb de comparaisons : **500 000**
  - nb d'affectations : **1 500** en moyenne